LAPIN AMK
Lapland University of Applied Sciences

OnkoKirjastoAuki

Aukioloaikasovellus Lapin korkeakoulukirjastolle

Anitta Örn

Opinnäytetyö
Teollisuuden ja luonnonvarojen osaamisala
Tietotekniikka
Insinööri (AMK)

2015

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Teollisuuden ja luonnonvarojen
osaamisala
Tietotekniikka

Tekijä	Anitta Örn	Vuosi	2015
Ohjaaja	Kenneth Karlsson		
Toimeksiantaja	Lapin korkeakoulukirjasto		
Työn nimi	OnkoKirjastoAuki : aukioloaikasovellus Lapin korkeakoulukirjastolle		
Sivu- ja liitemäärä	45 + 1		

Opinnäytetyössä perehdyttiin iOS-ohjelmointiin toteuttamalla pienimuotoinen ohjelmointityö Lapin korkeakoulukirjastolle. Sovelluksen avulla käyttäjä voi tarkistaa kampuskirjastojen aukioloajat ja muutokset aukioloajoissa. Sovelluksesta kehitettiin protoilumenetelmällä esittelyversio, jonka toiminnallisuuksista kerättiin palautetta kirjaston henkilökunnalta ja opiskelijoilta. Palautteen avulla sovelluksen käyttöliittymää kehitettiin edelleen.

Opinnäytetyössä ohjelmointiprojektista toteutettiin projektin aloitus (idean kehittely) ja tarkennus (protoilu) vaiheet. Lopullisen sovelluksen ohjelmointi ja käyttöönotto tapahtuu sovelluksen jatkokehitysvaiheessa. Sovelluksen toteutuksessa on huomioitava kustannustehokkuus ja ylläpito. Mobiilisovellusten kehitystyössä pirstaloituminen on suuri ongelma. Pirstaloitumisella tarkoitetaan tilannetta, jossa sovellusta ei ole mahdollista suunnitella vain yhdesti ja ajaa sen jälkeen kaikilla laitteilla. Pirstaloitumista voidaan vähentää suosimalla kehitystyössä monialustaisia kehitystyökaluja tai web-teknologiaa. Koska aukioloaikasovelluksessa pääpaino on grafiikan sijaan toiminnallisuudessa, on kustannustehokkuuden ja laiteriippumattomuuden vuoksi harkittava, voidaanko aukioloaikasovellusta julkaista web-sovelluksena. Web-sovellusta suunniteltaessa on huomioitava ympäristön rajoitukset, esim. käyttöliittymässä ei voida toteuttaa samoja toiminnallisuuksia kuin natiivisovelluksessa.

Opinnäytetyössä toteutettua esittelysovellusta voidaan käyttää suunnittelun pohjana ja testiympäristönä, kun kirjastolla selvitetään millaisia toiminnallisuuksia mobiilisovellukselta halutaan ja mitä toimenpiteitä mobiilisovelluksen käyttöönotto vaatii kirjastolta.

Avainsanat iOS, ohjelmointi, kirjastopalvelut

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

School of Industry and natural resources
Degree programme of information technology

Author Anitta Örn Year 2015
Supervisor(s) Kenneth Karlsson
Commissioned by Lapland University Consortium Library
Subject of thesis IsLibraryOpen – library opening hours mobile application
Number of pages 45 + 1

Object of the thesis was to produce a prototype of the mobile application for the Lapland University Consortia Library. With the application users are able to check library opening hours and if there were any changes to them. Prototype was used to collect user feedback from the library staff and students. Based on the feedback the application user interface was developed further.

Thesis covered the first two stages of the software development project. The programming of the final version of the application and implementation are carried out in the later stages. When developing the application, attention must be paid to cost-efficiency and maintenance. Fragmentation is a major problem in the mobile software development. Fragmentation means that the code cannot be written once and then run in every device. Multi-platform development tools can be used to minimize fragmentation or software can be designed to run in a web browser (a web app). Because the library app designed in this thesis does not use much graphics, it would be cost-efficient to publish it as a web application.

Project gave experience on software development and prototype can be used as a testing ground when developing mobile applications for the library.

Key words iOS, programming, library services

SISÄLLYS

TERMISTÖ

suunnittelumalli	(design pattern) on käyttöliittymäsovelluksen suunnitteluun kehitetty kokoelma valmiita käytäntöjä ja toteutustapoja. Suunnittelumallissa ideana on tarjota uudelleenkäytettävä ratkaisu useasti esiintyvään ongelmaan. Suunnittelumalli on korkea tason suunnitelma, eikä tarjoa valmiita koodiratkaisuja. (Lehtinen 2011.)
segue	siirtymä kahden näkymäohjaimen (view controller) välillä (Apple 2012a).
sovelluskehys	(framework) luokkakirjasto, josta periyttämällä erikoistetaan sovelluksessa käytettävät luokat. Kehys määrittelee perusrakenteen, jonka puitteissa ohjelmoijan on toimittava. Poikkeaa siten perinteisestä aliohjelmakirjastosta. (Rintala & Jokinen 2013, 199-200.)
näkymäohjain	(view controller) hallinnoi sovelluksessa käytettyjä näkymiä (views). Näkymäohjaimet myös viestivät keskenään. Tämä tekee sovelluksen käyttöliittymän siirtymätilanteista jouhevia. Käyttöliittymäsuunnittelualusta (storyboard) sisältää valmiita näkymäohjainolioita, joiden avulla käyttöliittymän rakentaminen on hallittua. (Apple 2012a.)

1 JOHDANTO

Neljän vuoden kuluttua yli puolet internetin käytöstä tapahtuu mobiililaitteilla, arvioi tutkimusyhtiö Gartner (Gartner 2014). Tilastokeskuksen tutkimuksen mukaan 49 prosentilla 16 – 78 -vuotiaista on käytössään älypuhelin. 25 – 34 -vuotiaiden keskuudessa luku alkaa hipoa jo 70 prosenttia. (Suomen virallinen tilasto 2012.) Kännykkä kulkee mukana kaikkialle: sillä luetaan sähköposti, päivitetään Facebook ja katsotaan videoklippejä pelaamisesta puhumattakaan. Vaikka laitevalmistajien ja käyttöjärjestelmien markkinaosuudet tulevaisuudessa tulisivat muuttumaan paljonkin, ihmisten asiointikäyttäytyminen on muuttunut pysyvästi. Asiat halutaan hoitaa sillä hetkellä ja juuri sillä laitteella, joka kulloinkin on käden ulottuvilla.

Kirjastot ovat huomioineet mobiilikäytön tarjoamalla mobiililaitteille optimoituja www-sivuja. Muun muassa Applen App Storessa on lisäksi ladattavissa kirjastosovelluksia, jotka on suunniteltu hyödyntämään erityisesti mobiilialitteiden ominaisuuksia (haku App Storesta 13.12.2014). Kirjastokortilla tunnistautumalla käyttäjä voi mobiililaitteellaan esimerkiksi lukea ja varata e-kirjoja (eBokBib), uusia lainoja ja tarkastella omia asiakastietoja (BIBSYS Min Side) tai perehtyä johonkin aihepiiriin (New York Public Library BibLion). Helsingin kaupunginkirjastolla oli vuoteen 2014 asti käytössä Helmet Taskukirjasto, jonka avulla asiakas pystyi muun muassa käyttämään kännykkäänsä kirjastokorttina ja lainaamaan kirjaston aineistoa suoraan kaverilta (Helsingin kaupunginkirjasto 2013). Lapin korkeakoulukirjaston Juolukka –tietokannasta on tarjolla mobiilikäyttöön optimoitu versio.

1.1 Lapin korkeakoulukirjaston saavutettavuus asiakkaiden näkökulmasta

Lapin yliopiston kirjasto, Kemi-Tornion ammattikorkeakoulun kirjasto ja Rovaniemen ammattikorkeakoulun kirjasto yhdistyivät Lapin korkeakoulukirjastoksi vuonna 2010. Yhteisen Juolukka-tietokannan kautta kirjaston käyttäjien ulottuville tulivat kirjaston kokoelmat kolmessa kaupungissa: Kemissä, Torniossa ja

Rovaniemellä. Vuonna 2014 kirjaston toimipisteitä on yhteensä seitsemän. Kirjaston käyttäjät voivat valita asiointipaikakseen minkä tahansa seitsemästä kirjastosta riippumatta asuinpaikkakunnasta tai missä korkeakoulussa he opiskelevat tai työskentelevät. Sujuvan asioinnin varmistamiseksi kirjaston käyttäjät tarvitsevat ajantasaisen tiedon siitä, miten kukin kirjasto on auki.

Vuonna 2014 Kemi-Tornion ammattikorkeakoulu ja Rovaniemen ammattikorkeakoulu yhdistyivät Lapin ammattikorkeakouluksi. Ammattikorkeakoulun kirjaston toimipisteissä tapahtui vuoden aikana isoja muutoksia. Rovaniemellä ammattikorkeakoulun kirjastot yhdistyivät yhdeksi Jokiväylän kirjastoksi. Kirjaston henkilökuntaan kohdistui vähennyksiä sekä Rovaniemellä, Kemissä ja Torniossa. Muutokset ovat heijastuneet erityisesti aukioloaikoihin. Kirjastot ovat olleet tavallista enemmän suljettuna ja aukioloaikoihin on tullut itsepalveluaikoja. Muutoksista on tiedotettu kirjaston www-sivulla, Facebookissa, kirjaston tiloissa ja korkeakoulun Avac -infokanavilla. Nämä viestintäkanavat eivät kuitenkaan ole onnistuneet tavoittamaan kirjaston käyttäjiä halutulla tavalla.

Ongelmalliseen aukioloaikatiedotukseen lähdettiin hakemaan ratkaisua mobiilisovelluksesta. Aukioloaikasovelluksia on käytössä esim. Groningen yliopistokirjastolla. Groningen yliopistokirjaston mobiilisovelluksella voi mm. tarkistaa kirjaston päivän aukioloajan ja asiakaspäätteiden käyttöasteen.

1.2 Kirjaston palvelut osana virtuaalikampusta

Lapin ammattikorkeakoulussa Virtuaalikampus on sateenvarjotermi, jonka alaisuudessa kehitetään muun muassa etä- ja verkko-opetusta (Pruikkonen 2014). Virtuaalikampus liitetäänkin usein virtuaaliopiskeluun, mutta se voi tarkoittaa myös muita asioita, esim. maksullinen kurssivälityspalvelu (Ranska). (Schreurs 2011) Tässä työssä virtuaalikampuksella tarkoitetaan opiskelua tukevia palveluita, jotka ovat tilasta ja ajasta riippumatta opiskelijoiden käytössä. Mobiilisovelluksissa virtuaalikampus yleensä näyttäytyy kokoelmana palveluita, jotka tavalla tai toisella liittyvät opiskeluun tai opiskelijaelämään. Kirjaston palveluista kam-

pussovelluksen kautta on tarjolla yleensä muutamia palveluita, esimerkiksi kirjaston kokoelmaluettelo, lainojen uusinta, kysy kirjastonhoitajalta -neuvontapalvelu tai vaikkapa mahdollisuus lukea e-aineistoja. App Storesta löytyvä BI Mobile on yksi esimerkki tällaisesta sovelluksesta. Virtuaalikampussovellukset sisältävät myös monenlaista muuta opiskelijaelämään liittyvää palvelua, esimerkiksi

- opastava kampuskartta (Hitta på LiU)
- kaverien paikantaminen (University of Bradford)
- osallistuminen online kursseille (Kaplan University for iPad)
- käytettyjen kurssikirjojen kirpputori (StudyBooks)
- opiskelijaradio (Leeds Student Radio)
- turvallisuusohjeistus (UniSafe)

(haku App Storessa 13.12.2014).

Mobiilisovelluksissa sisältö painottuu palvelujen käyttöön. Tekstipainotteiden informaatio tarjotaan useimmiten www-sivuilla. Jos käyttäjä haluaa tarkempia tietoja esimerkiksi kirjastosta, mobiilisovellus ohjaa hänet www-sivulle. Näin tapahtuu esimerkiksi Tukholman yliopiston mobiilisovelluksessa (Stockholm University Guide).

2 TOIMEKSIANTO JA TYÖN RAJAUS

Opinnäytetyön tavoitteena on tutustua iOS-ohjelmointiin ja toteuttaa demosovellus Lapin korkeakoulukirjastolle. Demosovellus toimii suunnittelun pohjana ja testiympäristönä, kun kirjastolla selvitetään millaisia toiminnallisuuksia mobiilisovellukselta halutaan ja mitä toimenpiteitä mobiilisovelluksen käyttöönotto vaatii kirjastolta.

Demosovellus toteutetaan ilman tietokantaa. Tietokantaa simuloidaan koodiin sisältyvillä testimuuttujilla. Sovelluksen koodi suunnitellaan siten, että palvelimella sijaitsevan tietokannan käyttöönotto on mahdollista seuraavassa kehitysvaiheessa.

3 OHJELMISTOKEHITYSMALLI JA OHJELMOINTIYMPÄRISTÖ

3.1 Sovelluksen kehittäminen protoilumenetelmällä

Protoilumenetelmässä tavoitteena on rakentaa ennalta määritellyistä ominaisuuksista riisuttu prototyyppi, jolla voidaan kuitenkin tutkia tarvittavissa määrin ohjelmiston käyttäytymistä. Evoluutioprototyyppi kehitetään vaiheittain valmiiksi tuotteeksi, kun taas pois heitettävä prototyyppiä käytetään järjestelmän mallintamiseen, jonka jälkeen varsinaisen tuotteen rakentaminen aloitetaan alusta. Prototyyppi toimii todellisen ohjelmiston tapaan, esimerkiksi rajapintafunktiot palauttavat todellista käyttöä kuvaavia arvoja, vaikka lopullisen tuotteen todellisia arvoja tuottavat ohjelmalliset osat prototyypistä puuttuisivatkin. (Haikala & Mikkonen 2011, 38.) Protoilussa ongelmaksi saattaa muodostua prototyypin loputon parantelu sekä asiakkaan näkemys, että prototyyppi on jo valmis järjestelmä, vaikka käytännössä valtaosa työstä on vielä tekemättä. Prototyypistä ei tästä syystä kannata tehdä kovin viimeistellyn tuntuista tai näköistä. (Haikala & Märijärvi 2004, 42 - 44.)

Evoluutioprototyyppi on kehitysmenetelmänä lähellä iteratiivista kehitystä. Niin kutsutuissa ketterissä (agile) menetelmissä iteraatiot (kehityssyklit) ovat lyhyitä. Ketterässä menetelmässä alussa määritellään vain karkealla tasolla järjestelmän arkkitehtuuri, ja itse tuotetta kehitetään moduuli kerrallaan. Jokaisen kehityssyklin päätteeksi asiakkaalle toimitetaan järjestelmän toimiva osa. Asiakas pääsee siten käyttämään järjestelmän toimivia osia nopeammin kuin perinteissä ohjelmistotuotantomallissa, jossa pyritään toimittamaan asiakkaalle kokonainen täysin toimiva järjestelmä. (Koskenniemi 2012, 80, Haikala & Mikkonen 2011, 38.) Ketterässä menetelmässä dokumentointi pidetään suunnitteluvaiheessa kevyenä. Asiakas on mukana suunnittelussa koko projektin ajan. (Koskenniemi 2012, 80, 82.)

3.2 Ohjelmointiympäristön valinnassa huomioitavia asioita

3.2.1 Mobiilin ohjelmistokehityksen pirstaloituminen

Pirstaloituminen (fragmentation) on suuri ongelma mobiilisovellusten kehitystyössä. Pirstaloitumisella tarkoitetaan tilannetta, jossa sovellusta ei ole mahdollista suunnitella vain yhdesti ja ajaa sen jälkeen kaikilla laitteilla (”write once and run anywhere”) (Rajapakse 2008). Pyrkiessään julkaisemaan sovelluksen mahdollisimman monelle alustalle tavoittaakseen mahdollisimman laajan käyttäjäkunnan, suunnittelija joutuu opettelemaan usean eri kehitysympäristön ja ohjelmointikielen. Tällainen kehitystyö on kallista. (Amatya 2013, 2, Charland & Leroux 2011.)

Pirstaloitumista voidaan vähentää suosimalla kehitystyössä monialustaisia (cross-platform, multiplatform) kehitystyökaluja. Mikkosen ja Taivalsaaren (2011) mukaan web-teknologiat kuten HTML5 ja WebGL mahdollistavat lähitulevaisuudessa yhä paremmin laiteriippumattomien sovellusten ohjelmoinnin suoraan webalustalle (Mikkonen & Taivalsaari 2011). Mobiililaitteet ovatkin tässä mielessä yhteneväisiä, että ne käyttävät yhteneväisiä selainstandardeja. Hybridi sovelluskehitys pyrkii yhdistämään natiivin sovelluskehityksen ja webalustan parhaat ominaisuudet. Hybridisovelluksessa webkoodi voi rajapintojen (API) välityksellä käyttää laitteen ominaisuuksia esim. kameraa. Hybridiin sovelluskehitykseen on tarjolla kehitysympäristöjä esim. Phonecap. (Amatya 2013, 12-13, Charland & Leroux 2011.)

3.2.2 Sovelluksen suunnittelu yhdelle alustalle

Natiivia, yhdelle alustalle toteutettua, sovelluskehitystä perustellaan usein suorituskyvyllä (käännetyn koodin suorittaminen on nopeampaa kuin koodin tulkkaaminen) ja hiotulla käyttökokemuksella (sovellus voi hyödyntää alustan kaikkia ominaisuuksia). Web teknologiat yltävät yhä paremmin natiivin sovelluksen suorituskykyyn, mutta käyttöliittymän toteutuksessa natiivi sovellus on edelleen

monipuolisempi. Sovelluskehittäjä joutuukin usein tekemään kompromisseja kustannusten ja käyttökokemuksen välillä. (Charland & Leroux 2011, Amatya 2013, 2.)

Valitsin ohjelmointympäristöksi Applen kehitysympäristön Xcoden, koska siinä on mahdollista käyttää visuaalista ohjelmointitapaa (storyboard) ja valmiita olioita esim. DatePicker. Toimivan sovelluksen toteuttaminen vaati siten suhteellisen vähän varsinaista koodin kirjoittamista. Xcode sopii tässä mielessä hyvin protoiluun.

4 SOVELLUKSEN SUUNNITTELU JA OHJELMOINTI

4.1 Käyttöliittymän suunnittelu

Suunnittelin sovelluksen käyttöliittymäluonnoksen mobiiliohjelmoinnin kurssilla. Esittelin luonnoksen toimeksiantajalle, jonka pohjalta sain hyväksynnän sovelluksen prototyypin toteutukseen. Käyttöliittymäluonnos toimi siten myös sovelluksen vaatimusmäärittelynä.

4.1.1 iOS -käyttöliittymäsuunnittelun periaatteet

Applen (2014a) käyttöliittymäsuunnitteluohjeet antavat suuntaviivoja, miten iOS laitteelle suunnitellun sovelluksen tulisi toimia, jotta sovelluksen käyttökokemus olisi iOS laitteelle tunnusomainen. iOS laitteelle suunnitellun sovelluksen tunnusomaisia piirteitä ovat esimerkiksi, että käyttöliittymä hyödyntää laitteen koko näyttöä, ulkoasu on ilmava ja siinä hyödynnetään tyhjää aluetta (negative space), yhteneväinen ilme saadaan aikaa avainväreillä ja iOS laitteelle suunnitelluilla fonteilla. Sisältö esitetään tasoissa (layers), joiden välillä käyttäjä liikkuu. (Apple 2014a)

Aukioloaikasovelluksen käyttökokemuksen suunnittelussa huomioitiin seuraavia ohjeita:

- Tärkein tieto näytetään heti kun sovellus aukeaa. Aukioloaikasovellus avautuu näkymään, jossa näytetään yhden kirjaston kuluvan päivän aukioloaika.
- Käyttäjän ei tarvitse määritellä asetuksia. Sovellus avaa aina viimeisimmäksi katsotun kirjaston aukioloaikatiedon.
- Sovellus avautuu laitteen oletusasentoon (pystyasento).
- Sovellus käyttää aloitusnäkymää (launch image).
- Sovellus sulkeutuu, kun käyttäjä siirtyy toiseen sovellukseen.

- Sovellus ilmoittaa, jos jokin toiminto ei ole päällä. Aukioloaikasovellusta ei voi käyttää, jos käyttäjällä ei ole internet yhteyttä tai ulkoisella palvelimella olevasta tietokannasta ei voida hakea tietoja. Näille tilanteille luodaan huomautusikkunat.
- Sovellus käyttää navigointipalkkia. Navigointipalkki kertoo käyttäjälle, mihin sovellukseen avoinna oleva näkymä kuuluu, ja tarjoaa käyttäjälle mahdollisuuden palata sovelluksen päänäkymään.
- Käyttäjä syöttää sovellukselle tietoja napauttamalla ruutua (tap). Kun käyttäjä on esim. valinnut napauttamalla listalta kirjaston, jonka päivän aukioloajan hän haluaa nähdä, sovellus siirtyy valintalistalta välittömästi päänäkymään, jossa tieto esitetään.
- Käyttäjän tekemä valintatieto tallennetaan heti sovelluksen muistiin.
- Sovellus käyttää iOS laitteen Ilmoituskeskusta (Notification Centre) ilmoittaakseen poikkeavista aukioloajoista.

Sovelluksen kuvake (app icon) hyödyntää kirjaston logoa. Kuvake kertoo käyttäjälle heti, että sovellus on kirjastoon liittyvä palvelu (kuva 1).

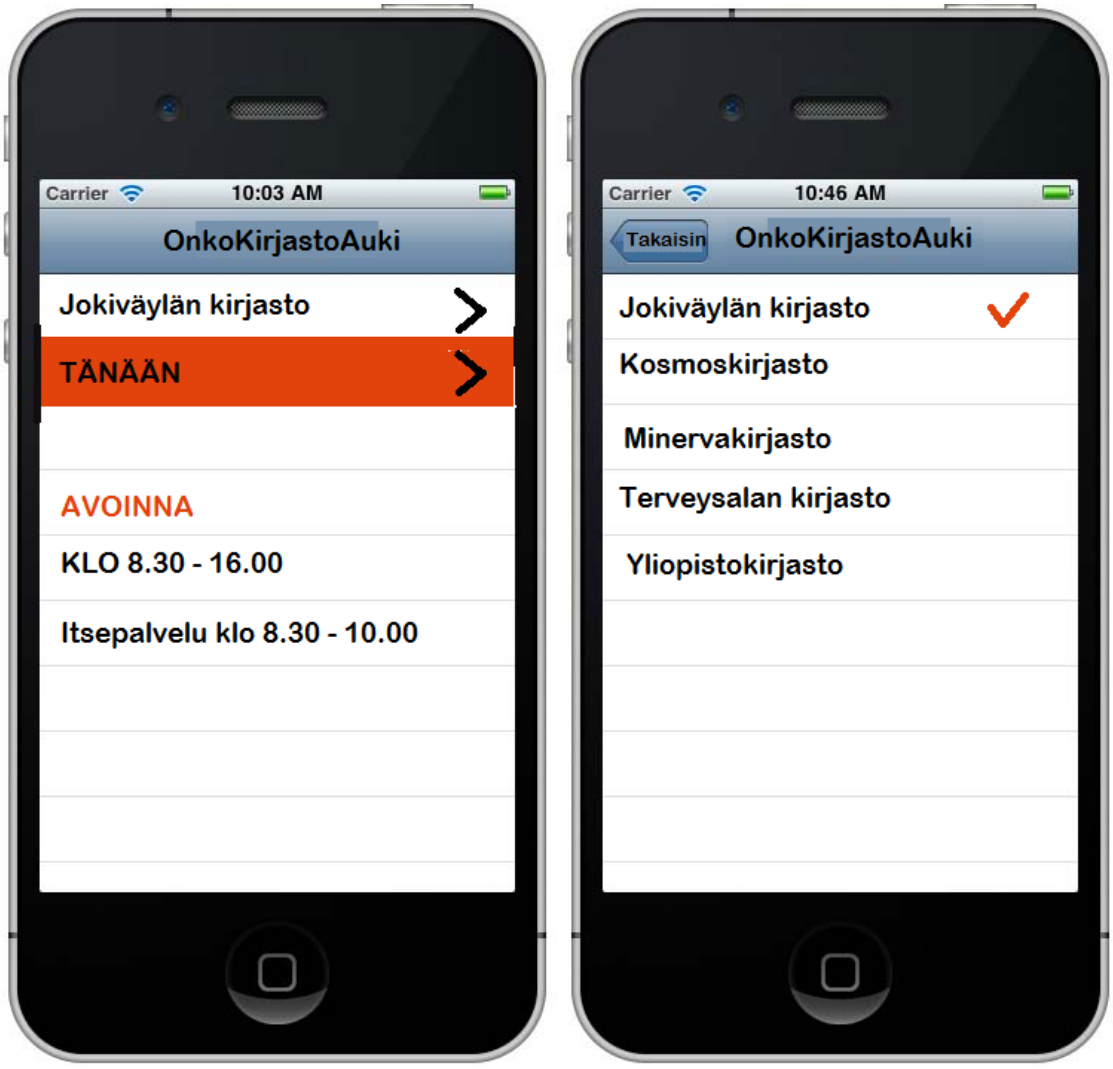
Kuva 1. Aukioloaikasovelluksen sovelluskuvake (app icon). Luonnos.

Sovelluksen latautuessa näytetään aloituskuva (launch image). Aloituskuva on staattinen kuva, joka on näkyvissä vain sen aikaa kun sovellus latautuu (kuva 2).

Kuva 2. Aloitusnäkymä (launch image). Luonnos.

4.1.2 Luonnokset käyttöliittymän näkymistä

Sovellus koostuu päänäkymästä ja valintanäkymästä. Päänäkymässä esitetään kirjaston kuluvan päivän aukioloaikatieto (kuva 3).



Kuva 3. Sovelluksen päänäkymä ja valintanäkymä. Kuvassa valintanäkymässä lista kirjastoista. Luonnos.

Päänäkymän Kirjaston nimi -palkki ja TÄNÄÄN -palkki aukaisevat napautettaessa valintanäkymän, joko listan kirjastoista tai kalenterinäkymän (kuva 3 ja kuva 4). Käyttäjän tekemän valinnan perusteella päänäkymän tekstikenttien AVOINNA ja KLO sisältö muuttuvat.

Kuva 4. Päänäkymä ja kalenterinäkymä. Luonnos.

Aukioloaikatiedot ovat haettavissa 6 kk:n ajalta. Sovellus ilmoittaa, jos haettua tietoa ei löydy tietokannasta (kuva 5).

Kuva 5. Huomautusikkuna. Luonnos.

Kuvien värit eivät vastaa lopullisen sovelluksen värejä. Kirjaston tunnusvärit ovat oranssi ja musta, joita käytetään kauttaaltaan sovelluksessa.

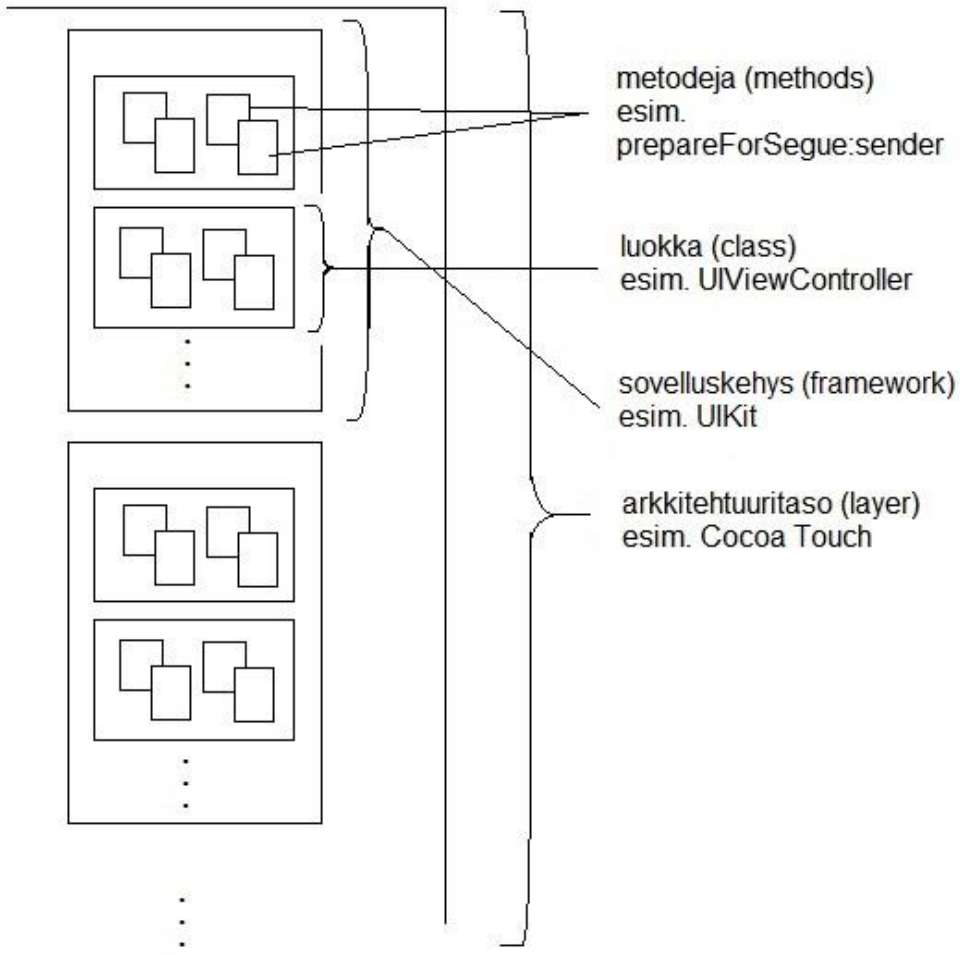
Ilmoituskeskuksen kautta lähetetyt ilmoitukset näkyvät laitteen määritysten mukaisesti (kuva 6).

Kuva 6. Ilmoituskeskuksen kautta lähetetty ilmoitus. Luonnos.

4.2 iOS –arkkitehtuuri ja suunnittelumallit

iOS –käyttöjärjestelmä rakentuu neljästä arkkitehtuuritasota: Cocoa Touch, Media, Core Services ja CoreOS. Cocoa Touch on ylin arkkitehtuuritaso, joka tarjoaa toiminnot esim. kosketustunnistukselle (gesture recognition). Cocoa Touch on rakennettu ModelViewController –suunnittelumallin pohjalle. (Khan 2015, 202-203.) MVC-suunnittelumallissa erotetaan näkymä sovelluslogiikasta eli järjestelmä jaetaan malliin ja näkymään. MVC-sovellusmallissa on lisäksi ohjain tai käsittelijä (controller), jonka vastuulla on näkymän ja mallin yhdistäminen. Mallin tehtävänä on tietomallin toteuttaminen, tiedon varastointi ja tiedon käsittelyyn liittyvien toimintojen toteuttaminen. Näkymä on käyttäjälle näkyvä järjestelmän osa. Näkymän tehtävänä on tarjota käyttäjälle työkalut, joilla järjestelmää voi hallita. Ohjain vastaa käyttäjän syötteiden hallinnasta. Näkymät ja ohjaimet muodostavat järjestelmän käyttöliittymän. Näkymän (laajemmin käyttöliittymän) erottaminen sovelluslogiikasta tuo hyötyjä esim. testausvaiheessa. Ohjelman sovelluslogiikka voidaan testata erillään käyttöliittymästä, ja päinvastoin. (Apple 2014, Lehtinen 2011.) Sovelluksessa käytetty Table View –näkymä on toimintaperiaatteeltaan MVC-suunnittelumallin mukainen.

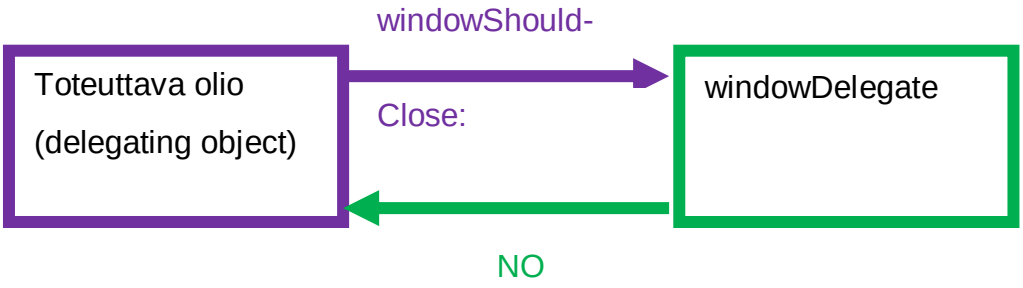
Arkkitehtuuritasojen sisältämät sovelluskehykset (frameworks) luovat kullekin tasolle rakenteen. Sovelluskehykset pitävät sisällään luokkia ja luokat metodeja. (Khan 2015, 132, 202-203.) Kuvassa 7 on havainnollistettu käsitteiden suhdetta toisiinsa.



Kuva 7. iOS arkkitehtuuri. Mukaillen Khan 2015, 132, 202-203.

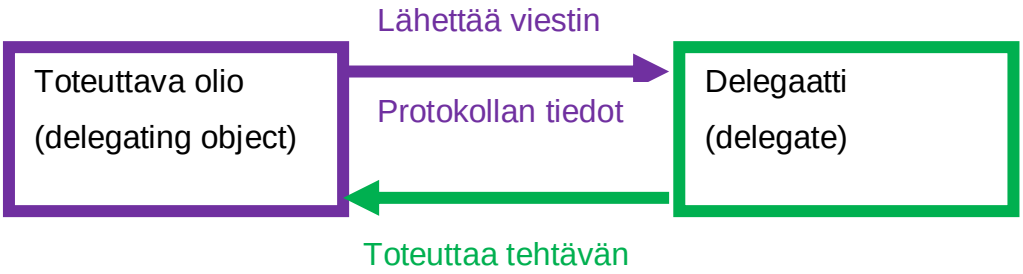
Tilanteissa, joissa tietoa siirrettiin kahden näkymän välillä, hyödynnettiin delegaatti -suunnittelumallia. Delegaatti (delegation) on malli, jossa olio toimii toisen olion puolesta tai yhteistyössä sen kanssa. Suunnittelumallissa delegaatti (delegate) on olio, joka saa viestin toteuttavalta oliolta (delegating object). Viesti kertoo delegaatille, että toteuttava olio on aikeissa toteuttaa toiminnon. Delegaatti reagoi viestiin esim. lähettämällä muuttujan arvon, joka vaikuttaa siihen,

miten toteuttava olio käsittelee toiminnon. (Apple 2015.) Kuvassa 8 on havainnollistettu delegaatin toimintaa tilanteessa, jossa ikkunaa ollaan sulkemassa.



Kuva 8. Delegaatti toiminnassa. (Apple 2015.)

Aukioloaikasovelluksessa delegaattina toimii näkymänhallintaolio (view controller). Ennen kuin olio pystyy käyttämään delegaattimallia tiedonvälitykseen, sen on määriteltävä protokolla. Protokolla sisältää metodimäärittelyt, jotka toimivat rajapintana kommunikoivien olioiden välillä. (Khan 2015, 291.)



Kuva 9. Delegaatin ja toteuttavan olion välinen viestintä. (Khan 2015, 291.)

Kuvassa 9 on kuvattu Delegaatin ja toteuttavan olion välinen viestintä.

4.3 Sovelluksen ohjelmoinnissa tehdyt ratkaisut

Xcode-ympäristössä käyttöliittymäsuunnittelua voidaan toteuttaa joko storyboardilla tai xib-tiedostoilla. Storyboard on toiminto, joka mahdollistaa työskentelyn useiden näkymien kanssa samanaikaisesti. Näkymät ja käyttöliittymän tarvitsemat muut toiminnot, kuten painikkeet ja tekstikentät, raahataan suunnittelualus-

talle (Interface Builder canvas) ja yhdistetään toisiinsa alustan tarjoamien työkalujen avulla. Storyboard esittää näkymien väliset siirtymät (segues) visuaalisesti, mikä auttaa hahmottamaan sovelluksen toimintoja kokonaisuutena. (Khan 2015, 236.)

4.3.1 Aloitusnäkymän ja kirjastolistausnäkymän toteutus

Aloitusnäkymä ja kirjastolistausnäkymä on toteutettu käyttäen Table View -näkymää. Table View on iOS –sovelluksissa käytetty käyttöliittymäolio (object), jonka avulla voidaan esittää tietoa selattavana listana. TableView on UITableView –luokan ilmentymä (instance). Table View rakentaa listan soluista, jotka ovat samalla rivejä. Solut ovat UITableViewCell -luokan olioita. Solut voidaan esittää kahdella tavalla: staattisena soluna tai dynaamisena prototyyppinä. Staattisia soluja käytetään, kun listanäkymässä halutaan olevan kiinteä määrä rivejä. Staattisten solujen ulkoasua voi myös muokata yksilöllisesti. Toteutettaessa Table View -näkymä, jonka rivimäärä muuttuu ohjelman suorituksen aikana, käytetään dynaamista prototyyppiä. Kun solu toimii dynaamisena prototyyppinä, sitä käytetään mallina muille näkymän soluille. Ulkoasullisesti solut ovat siten toistensa kopioita. UIKit -sovelluskehys määrittelee solujen ulkoasutyylit. (Apple 2013.)

UITableView –luokka hallinnoi tiedon esittämistä ja syöttöä. Toimiakseen se tarvitsee avukseen delegaatti- ja data source -oliot. Delegaatti sisältää metodit ulkoasun hallinnointiin ja käyttäjän toimintoihin, esim. metodi, joka seuraa, onko käyttäjä napauttanut solua (tableView:didSelectRowAtIndexPath). Data source hallinnoi tiedon esittämiseen liittyviä metodeja, esim. tableView:cellForRowAtIndexPath palauttaa yksittäisen solun polkutunnisteen. Solut yksilöidään polkutunnisteella (index path). Polku rakennetaan NSIndexPath –luokan avulla. NSIndexPath kuuluu Foundation -sovelluskehykseen. UIKit -sovelluskehys liittää Foundation –sovelluskehyksen sisältämään NSIndexPath -luokkaan ohjelmointirajapinnat, jotta UITableView –luokan oliot voivat yksilöidä listanäkymän rivit (rows). (Apple 2013, Apple 2012b, Feiler 2013, 230-231.)

UITableView -olion, delegaatin ja data sourcen toimintaa ohjaa TableViewController -olio. Apple suosittelee käytettäväksi UITableViewController -luokkaa Table View -näkymien rakentamissa. Ohjelman suorituksen aikana UITableViewController luo näkymän ja asettaa samalla itsensä delgaatiksi ja data sourceksi. (Apple 2013, Feiler 2013, 231.) Tämän vuoksi näitä ei tarvitse erikseen määritellä header –tiedostossa.

Aloitusnäkymässä Table View toimii ulkoasua yhtenäistävänä rakenteena. Solut ovat staattisia ja ne ovat ryhmitelty kahteen osaan. Käytettäessä storyboardia, staattiset solut ja solujen ryhmittelyn saa määritettyä Attributes Inspector -työkalulla. Jokaiseen soluun lisätään tekstikenttä (UILabel), jota käytetään tekstin esittämiseen, esim. valittu kirjaston nimi. Storyboardia käytettäessä aloitusnäkymän rakentaminen ei vaatinut varsinaista koodin kirjoittamista.

Kirjastolistausnäkymässä TableView -oliota käytetään dynaamisen listan luomiseen. Listan solujen tiedot haetaan taulukosta nimeltä kirjastot.

```
// KirjastolistaTableViewController.m

- (void)viewDidLoad
{
    [super viewDidLoad];
    // luodaan kirjastot taulukon sisältö
    kirjastot = [[NSMutableArray alloc]init];
    [kirjastot addObject:@"Jokiväylä"];
    [kirjastot addObject:@"Kosmoskirjasto"];
    [kirjastot addObject:@"Minervakirjasto"];
    [kirjastot addObject:@"Terveysalan kirjasto"];
    [kirjastot addObject:@"Yliopisto"];

}
```

Taulukko voidaan toteuttaa myös plist -tiedostoon. Property List (plist) tallettaa tiedot erilliseen tiedostoon, josta ne haetaan. Tämä tekee koodista siistimpää, mutta plist ei sovi isojen tietomäärien esittämiseen. Jos tietomäärät ovat suuria

ja niitä tarvitsee muokata useasti, on tietokanta silloin parempi vaihtoehto tiedon tallennukseen. (Khan 2015, 446.)

Dynaamisen listan luominen aloitetaan määrittelemällä rivit ja sarakkeet.

```
// KirjastolistaTableViewController.m

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // sarakkeita on yksi.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView number-
OfRowsInSection:(NSInteger)section
{
    // rivejä on yhtä paljon kuin kirjastot –taulukossa nimiä
    return [kirjastot count];
}
```

Kirjastot –taulukon sisältö lisätään soluihin.

```
// KirjastolistaTableViewController.m

- (UITableViewCell *)tableView:(UITableView *)tableView cellFor-
RowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"valitseKirjastoCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdenti-
fier:CellIdentifier forIndexPath:indexPath];
    if (cell == nil)
    {
        cell = [[UITableViewCell alloc] initWith-
Style:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier];
    }

    cell.textLabel.text = [kirjastot objectAtIndex:indexPath.row];

    return cell;
}
```

Jotta solun tunnistepolku voidaan tallettaa muuttujaan ja esim. lähettää toiselle näkymälle, pitää ottaa käyttöön UITableViewCell –luokka.

TableViewCell –oliossa määritetään metodi.

```
//  valitseKirjastoTableViewCell.m

#import "valitseKirjastoTableViewCell.h"

@implementation valitseKirjastoTableViewCell
@synthesize kirjastoNimi;

- (id)initKirjastoNimi:(NSString *) valinta
{
    self.kirjastoNimi = valinta;
    return self;
}
```

Metodia käytetään tunnistepolun tallentamiseen.

```
// KirjastolistaTableViewController.m

#import "KirjastolistaViewController.h"
#import "valitseKirjastoTableViewCell.h"

- (void)tableView:(UITableView *)tableView didSelectRowAtIndex-
Path:(NSIndexPath *)indexPath{

    valitseKirjastoTableViewCell *solu = (valitseKirjastoTableView-
Cell*)[tableView cellForRowAtIndexPath:indexPath];
   // muuttuja nimeltä solu sisältää tiedon polusta, jota voidaan käyttää
solun tekstisisällön esittämiseen

    NSLog(@"solun teksti...%@",solu.textLabel.text);
   // tai solun tekstisisällön välittämiseen toiselle näkymälle

    [self.delegate valintaTehty: solu.textLabel.text];
}
```

4.3.2 Tietojen siirto näkymien välillä

Jotta aloitusnäkymässä voidaan näyttää esim. käyttäjän kirjastolistalla tekemä valinta, täytyy tiedon siirtyä kirjastolistanäkymästä aloitusnäkymälle. Tehokkain tapa toteuttaa tällainen tiedonsiirto on käyttää delegaattia (Khan 2015, 291). Kuvassa 10 on havainnollistettu delegaatin ja protokollan toimintaa, esimerkkinä aloitusnäkymän ja kirjastolistanäkymän välinen tiedonsiirto.

Aloitusnäkymän header –tiedostossa määritellään, minkä näkymien delegaattina aloitusnäkymä toimii. Tässä tapauksessa toteuttavina olioina (delegating objects) toimivat kirjastolistanäkymä ja kalenterinäkymä.

```
// OnkoAukiTableViewController.h

#import <UIKit/UIKit.h>
#import "KirjastolistaTableViewController.h"
#import "KalenteriViewController.h"

@interface OnkoAukiTableViewController : UITableViewController
<kirjastolistaTableViewControllerDelegate,
kalenteriViewControllerDelegate>
```

Siirtymä näkymien välillä tapahtuu seguen avulla. Siirtymän yhteydessä toteuttavalle näkymälle kerrotaan, että aloitusnäkymä on delegaatti.

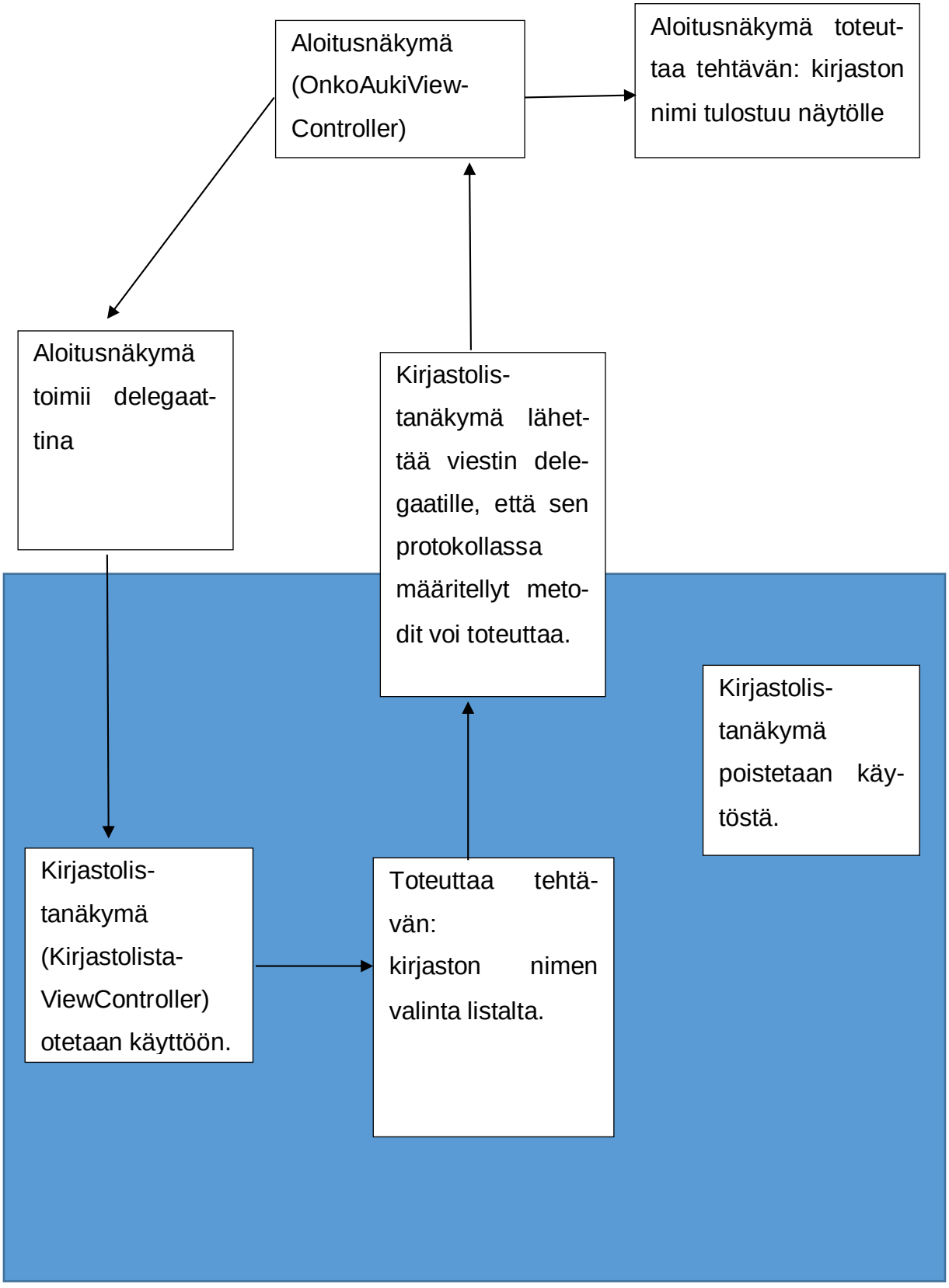
```
// OnkoAukiTableViewController.m

-(void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{
    if([segue.identifier isEqualToString:@"valitseKirjasto"]){
        KirjastolistaTableViewController *kirjastolistaTableViewController
= segue.destinationViewController;

        // kirjastolistanäkymälle tieto että aloitusnäkymä on delegaatti
        kirjastolistaTableViewController.delegate = self;
    }
    else if ([segue.identifier isEqualToString:@"valitsePvm"]){
        KalenteriViewController *kalenteriViewController = se-
gue.destinationViewController;
        // kalenterinäkymälle tieto että aloitusnäkymä on delegaatti
```

```
    kalenteriViewController.delegate = self;
}
```

Jotta delegatti ja toteuttava näkymä pystyvät kommunikoimaan keskenään, täytyy toteuttavalle näkymälle määritellä protokolla. Protokolla sisältää metodit, joissa määritellään, mitä toimintoja delegaatti haluaa toteuttavan näkymän tekevän. (Khan 2015, 291.)

```
// KirjastolistaTableViewController.h

#import <UIKit/UIKit.h>

@protocol kirjastolistaTableViewControllerDelegate <NSObject>

-(void) valintaTehty: (NSString *) name;

@end

@interface KirjastolistaTableViewController : UITableViewController

@property (nonatomic, assign) id <kirjastolistaTableViewControllerDelegate> delegate;

@end
```

Protokollan lisäksi toteuttavan näkymän header -tiedostossa määritellään käytettävä delegaatti.



Kuva 10. Delegaatti ja protokollan toiminta. Mukaillen Khan 2015, 292.

Delegaatin avulla tieto siirretään toteuttavalta näkymältä aloitusnäkymälle.

```
// KirjastolistaTableViewController.m

- (void)tableView:(UITableView *)tableView didSelectRowAtIndex-
Path:(NSIndexPath *)indexPath{

    valitseKirjastoTableViewCell *solu = (valitseKirjastoTableView-
Cell*)[tableView cellForRowAtIndexPath:indexPath];
   // solun sisältö lähetetään aloitusnäkymän metodille valintaTehty
    [self.delegate valintaTehty: solu.textLabel.text];
}
```

Aloitusnäkymä suorittaa toteuttavan näkymän lähettämän tiedon perusteella protokollassa määritellyn metodin.

```
// OnkoAukiTableViewController.m

-(void)valintaTehty: (NSString *) name{
    [self dismissViewControllerAnimated:YES completion:nil];
    kirjastoNimi.text = name;
}
```

Tässä tapauksessa kirjastolistanäkymän solun sisältämä teksti tulostuu aloitusnäkymän kirjastoNimi -tekstikenttään.

4.3.3 Kalenterinäkymän toteutus

Alustavassa käyttöliittymäluonnoksessa kalenteri oli toteutettu perinteisellä kuukausikalenterinäkymällä. Apple ei kuitenkaan tarjoa tämäntapaista valmista kalenterioliota. Perinteisen kalenterinäkymän sijaan Apple on toteuttanut päivämäärävalintaan DatePicker -toiminnon. Perinteisen kalenterinäkymän toteutukseen on mahdollista käyttää valmiita kolmannen osapuolen vapaan lähdekoodin kalenterisovelluksia.

Storyboardia käytettäessä DatePicker –olion voi valita kirjastosta (Object Library) ja raahata näkymään. Xcode luo automaattisesti koodin, jota muokkaamalla voi vaikuttaa esim. päivämäärän esitystapaan.

```
// KalenteriViewController.h

#import <UIKit/UIKit.h>
@protocol kalenteriViewControllerDelegate <NSObject>

-(void) tehtyPvm: (NSString *) aika;

@end


@interface KalenteriViewController : UIViewController

@property (nonatomic, assign) id <kalenteriViewControllerDelegate>
delegate;
@property (nonatomic, retain) NSString * pvmvalinta;

- (IBAction)valittu:(id)sender;

@property (weak, nonatomic) IBOutlet UILabel *pvm;
@property (weak, nonatomic) IBOutlet UIDatePicker *pvmValinta;

@end



// KalenteriViewController.m
- (void)viewDidLoad
{
    [super viewDidLoad];
    [self.pvmValinta addTarget:self
action:@selector(updateBtnTextFromPicker:) forControlEv-
ents:UIControlEventValueChanged];
}

}
- (IBAction)updateBtnTextFromPicker:(id)sender {

    NSDate *date = [self.pvmValinta date];
    NSLog(@"Valitsit päivän %@", [date description]);
    // muutetaan päivämäärän esitystapaa
    NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
    [dateFormatter setDateFormat:@"dd.MM.yyyy"];
    NSString *formatedDate = [dateFormatter stringFromDa-
te:self.pvmValinta.date];
```

```
    / päivämäärätieto näytetään tekstikentässä nimeltä pvm
    self.pvm.text=formatedDate;
    päivämäärätieto tallennetaan muuttujaan pvmvalinta, jota käytetään
napinpainallus-metodissa
    pvmvalinta =self.pvm.text;
}

// päivämäärätieto lähetetään aloitusnäkymälle kun on painettu OK
nappia
- (IBAction)valittu:(id)sender {
    [self.delegate tehtyPvm: pvmvalinta];

    // samalla kalenterinäkymä poistetaan näkymähierarkia -taulukosta,
jolloin palataan automaattisesti aloitusnäkymään
    NSMutableArray *viewHeirarchy =[[NSMutableArray alloc] init-
WithArray:[self.navigationController viewControllers]];
    [viewHeirarchy removeObject:self];
    self.navigationController.viewControllers = viewHeirarchy;

}
@end
```

4.3.4 Tietojen lähettäminen palvelimelle

Käyttäjän valitsemat kirjaston nimi ja päivämäärätiedot lähetetään www-sivulle, jossa tietoja käytetään tietokantahaun tekemiseen. NSURLConnection -luokka tarjoaa metodit tiedon siirtoon www-yhteydellä. Lisäksi käytetään NSURL -luokkaa url-osoitetiedon käsittelyyn ja NSURLRequest -luokkaa yhteyden määrittelyyn. Jotta NSURLRequest -olion metodeja voidaan muokata, pitää käyttää NSMutableURLRequest -luokkaa. (Cox & Jones & Szumski 2012.)

```
    //lähettää kirjaston nimen www-sivulle

    NSString * kirjastoNimi = solu.textLabel.text;

    NSString *queryString = [NSString stringWithFormat:
    @"http://localhost:8888/helloP.php?name=%@", kirjastoNimi];
    NSMutableURLRequest *theRequest=[NSMutableURLRequest
    requestWithURL:[NSURL URLWithString: queryString]
```

```
    cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInter-
val:60.0];
   [theRequest setHTTPMethod:@"POST"];
   NSString *post = [NSString stringWithFormat:@"postedValue=%@",
   kirjastoNimi];
   NSData *postData = [post dataUsingEncoding:
   NSUTF8StringEncoding];
   [theRequest setHTTPBody:postData];

   NSURLConnection *con = [[NSURLConnection alloc]
   initWithRequest:theRequest delegate:self];
   if (con) {
      _receivedData=[NSMutableData data];
   } else {
      //virheilmoitus
   }

}
```

Käytettäessä asynkronista yhteyttä, NSURLConnection -olio kutsuu delegaattia, joka käsittelee tulevan datan ja mm. vastaa virhetilanteisiin (Cox & Jones & Szumski 2012, 38). Jotta näkymä pystyy vastaanottamaan delegaatin viestejä, header -tiedostossa tulee ottaa käyttöön NSURLConnectionDelegate (Risner 2013).

```
//  KirjastolistaTableViewController.h
#import <UIKit/UIKit.h>

@interface KirjastolistaTableViewController : UITableViewController
<NSURLConnectionDelegate> {
   NSMutableData* _receivedData;
   id delegate;
}
```

NSURLConnectionDelegate metodit käsittelevät datan siirtoa.

```
//metodia kutsutaan kun www-sivu vastaa pyyntöön

-(void)connection:(NSConnection*)conn didReceiveResponse:
(NSURLResponse *) response
{
   if (_receivedData == NULL) {
      _receivedData = [[NSMutableData alloc] init];
   }
```

```
    [_receivedData setLength:0];
    NSLog(@"didReceiveResponse: responseData length:(%lu)",
    (unsigned long)_receivedData.length);
}

//metodia kutsutaan kun www-sivun lähettämä data vastaanotettiin

- (void)connection:(NSURLConnection *)connection
didReceiveData:(NSData *)data {
    [_receivedData appendData:data];
}

//metodia käytetään kun tapahtuu yhteydenotossa tapahtuu virhetilanne

- (void)connection:(NSURLConnection *)connection
  didFailWithError:(NSError *)error {
    NSLog(@"Connection failed! Error - %@ %@",
        [error localizedDescription],
        [[error userInfo] objectFor-
Key:NSURLErrorFailingURLStringErrorKey]);
}
```

Tietokannasta haetut aukioloaikatiedot lähetetään www-sivun kautta sovellukselle.

```
// metodia kutsutaan kun datan vastaanotto on päättynyt

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
    NSLog(@"Succeeded! Received %lu bytes of data",(unsigned long)
    [_receivedData length]);

    NSString *responseText = [[NSString alloc]
    initWithData:_receivedData encoding: NSASCIIStringEncoding];

// tulostaa www-sivun sisällön kloAika kohtaan

    NSString *kloAika = responseText;

    [self.delegate tehty:kloAika];
}
```

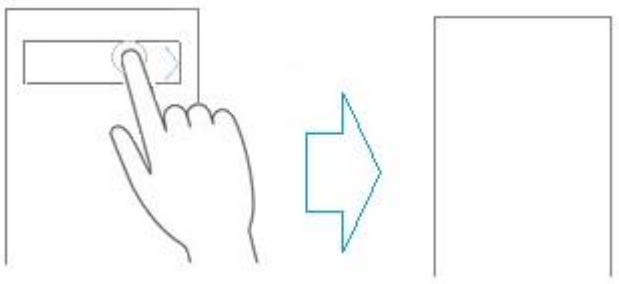
Tiedonsiirtoyhteyttä www-sivulle testattiin yksinkertaisella POST -metodilla, mutta lopullinen tiedonsiirtokoodi suunnitellaan aukioloaikatietokannan suunnittelun yhteydessä. Tietokannan toteutus ja palvelimen käyttöönotto toteutetaan

projektin jatkokehitysvaiheessa. Käyttöliittymän testausta varten tietojen hakua palvelimelta simuloitiin kevyellä if-ehtolauseella.

5 KÄYTTÖLIITTYMÄN TESTAUS

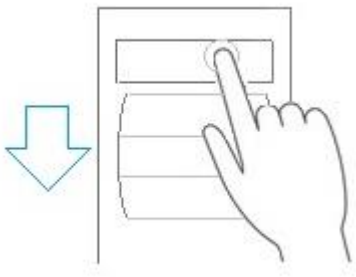
Käyttöliittymän testauksella kartoitettiin käyttäjäkokemusta ja itseohjautuvuutta, miten hyvin käyttäjä pystyy löytämään haluamansa toiminnot ilman ohjausta, intuitiivisesti. Käyttäjäkokemus (user experience) on sovelluksen suorituskyvyn ohella yksi tärkeimmistä tekijöistä, jotka vaikuttavat siihen, kuinka mielellään sovellus otetaan käyttöön. Mobiilisovelluksen käyttöliittymän suunnittelu käyttäjäkokemuksen näkökulmasta voidaan jakaa kahteen osaan: missä yhteydessä sovellusta käytetään (context) ja toteutusratkaisut (implementation). Siihen, missä ympäristössä sovellusta käytetään, ei voida vaikuttaa ohjelman suunnitteluvaiheessa, mutta ympäristön reunaehdoista tulee olla silti tietoinen. Sovellusta voidaan käyttää esim. erilaisilla laitteen versioilla. Sovelluksen suunnittelussa voidaan sen sijaan vaikuttaa toteutusratkaisuihin, kuten suorituskykyyn, ulkoasuun ja siihen, miten sovellus kommunikoi laitteen tarjoamien palveluiden kanssa. Käyttöliittymäsuunnittelua hankaloittaa esim. laitteiden erilaiset linjaukset käyttöliittymätoteutuksissa, esim. www-selaimen paluu-painike on eri laitteissa eri paikoissa. (Charland & Leroux 2011.)

Esittelin testaustilanteessa kolme käyttöliittymätoteutusta. Vertailusovelluksina olivat Applen Datecell –sovellus (Apple 2014b), jonka sisältöä muokkasin aukioloaikasovellukseen sopivaksi, ja Groningen yliopistokirjaston sovellus. Näiden käyttöliittymäratkaisut poikkesivat Lapin korkeakoulukirjastolle suunnitellusta sovelluksesta. Lapin korkeakoulukirjaston sovelluksessa käyttäjä siirtyy näkymien välillä napauttamalla valikkoa (kuva 11). Siirtyminen uuteen näkymään osoitetaan väkäsellä (disclosure indicator).



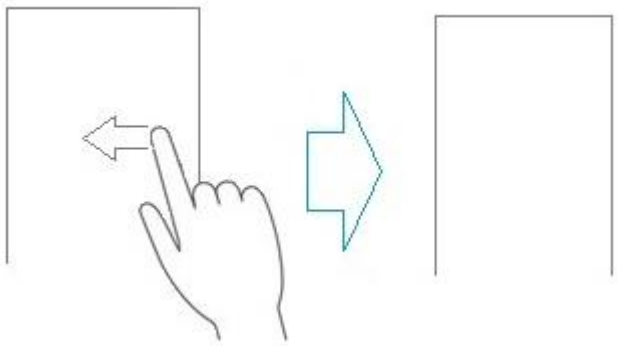
Kuva 11. Käyttöliittymä 1: Napautus vaihtaa näkymää.

Datecell -sovelluksessa toiminnot tapahtuvat yhdellä näkymällä. Valintarulla tulee näkyviin ja piilotetaan napauttamalla (kuva 12). Myös tässä sovelluksessa valintamahdollisuus osoitetaan väkäsellä.



Kuva 12. Käyttöliittymä 2: Napautus avaa ja piilottaa valintarullan (PickerView).

Groningen yliopistokirjaston sovelluksessa käyttäjä selaa kirjaston tietoja pyyhkäisemällä (kuva 13). Myös tässä sovelluksessa käyttäjällä on mahdollisuus siirtyä näkymien välillä käyttäen väkäsellä merkittyä painiketta.



Kuva 13. Käyttöliittymä 3: Pyyhkäisy vaihtaa näkymää.

Keräsin palautetta kirjastohenkilökunnalta ryhmähaastatteluin. Toteutin yhteensä kolme ryhmähaastattelua amk- ja yliopistokirjaston henkilökunnalle. Haastattelin myös yksittäisiä kirjastoasiakkaita Kemissä, Rovaniemellä ja Torniossa. Asiakkaat olivat amk- ja yliopisto-opiskelijoita.

Käyttöliittymien 1 ja 2 keskeneräisyys hankaloitti jonkin verran palautteen keruuta. Selkeitä ongelmakohtia olivat esim. käyttöliittymän 1 kalenterinäkymä, jossa käyttäjän tekemän valinnan jälkeen palattiin automaattisesti aloitusnäkymään. Käyttäjälle ei jäänyt aikaa muokata valintaansa. Päivämäärän valinta rullavalikolla koettiin toimivaksi, mutta viikonpäivien puuttuminen oli iso miinus. Tästä syystä perinteinen kalenterinäkymä koettiin paremmaksi. Käyttöliittymät 1 ja 2 koettiin selkeiksi. Groningen yliopistokirjaston sovellus sisälsi isoja kuvia, jotka koettiin häiritseviksi. Ydintoiminnot hukkuivat kuvainformaation sekaan. Väkänen osoitti yksiselitteisesti kohdat, joissa oli mahdollista tehdä valintoja (vaihtaa päivämäärää tai kirjastoa). Siirtyminen pyyhkäisemällä oli miellyttävä toiminto, mutta ei oleellinen tämän tapaisessa sovelluksessa. Käyttäjät korostivat toimintojen selkeyttä, kuvat ja grafiikka eivät olleet niin tärkeitä.

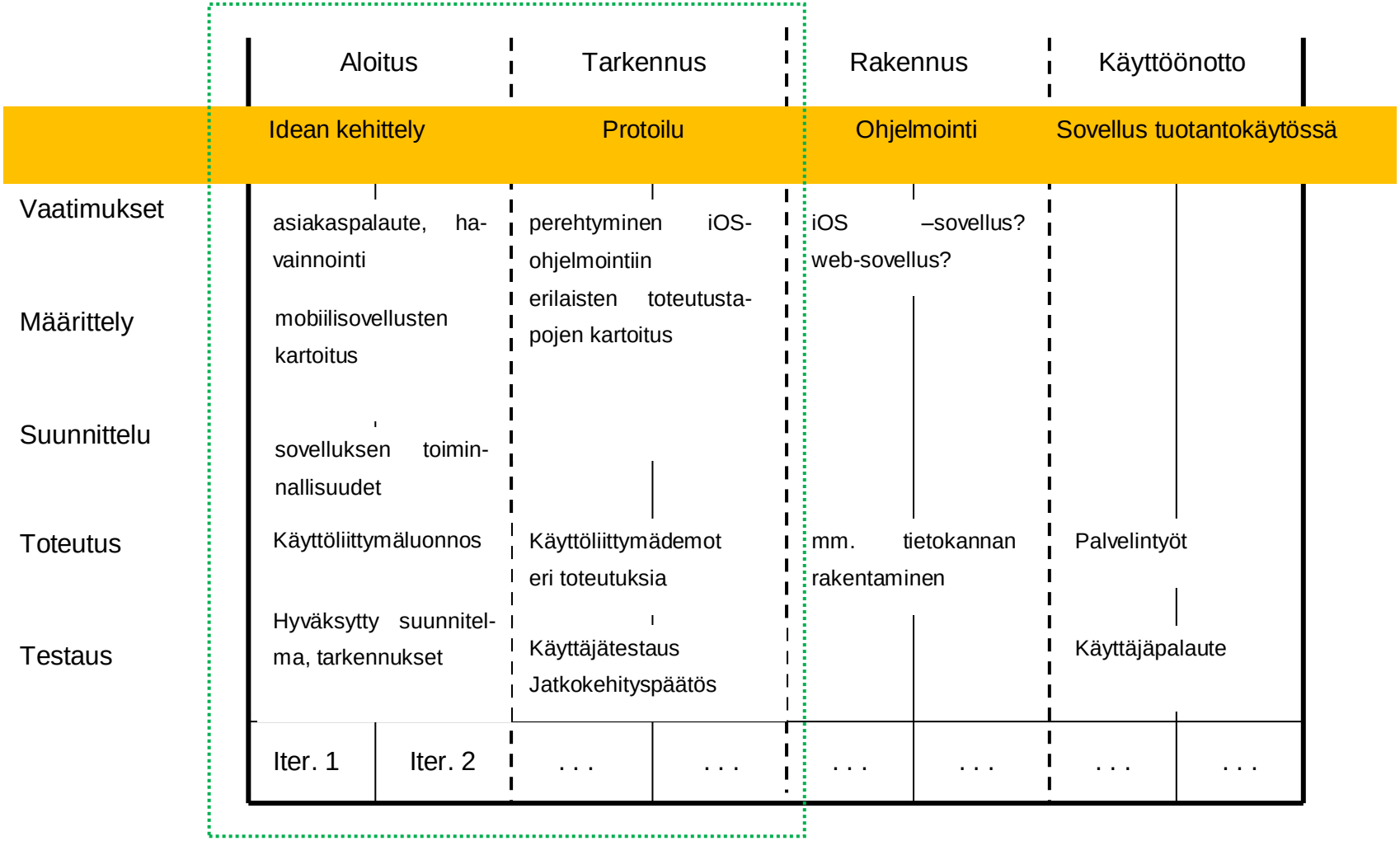
Groningen yliopistokirjaston sovelluksessa pystyi aukioloaikatietojen lisäksi tarkistamaan asiakaspäätteiden käyttöasteen. Tämä koettiin hyvänä toimintona ja tuovan sovellukselle lisäarvoa. Aukioloaikatieto yksinään ei ehkä riittäisi perustelemaan sovelluksen käyttöä, vaan sen rinnalla tulisi tarjota muitakin palveluita. Kaiken kaikkiaan käyttöliittymä 1 sai kannatusta selkeyden ja tutun toimintatavan vuoksi, mutta käyttäjät esittivät myös kaikkien kolmen käyttöliittymän piirteiden yhdistämistä. Esim. Groningen yliopistokirjaston sovelluksessa päivämäärävalinta -näkymässä näkyi yhdellä silmäyksellä kaikkien kirjastojen aukioloajat valitulta päivältä. Käyttäjien antaman palautteen pohjalta uusi käyttöliittymä voisi olla kuvan 14 mukainen.

Kuva 14. Käyttöliittymäehdotus käyttäjien antaman palautteen pohjalta. Luonnos.

Käyttöliittymäehdotuksessa kalenterinauha tulee näkyviin napauttamalla TÄNÄÄN -painiketta. Päivämäärävalinta hyväksytään OK -painikkeella, jonka jälkeen aukioloaikatieto päivittyy kaikkien listan kirjastojen kohdalla. Valittu päivämäärä näytetään TÄNÄÄN -tekstin tilalla.

6 JATKOKEHITYS

Opinnäytetyö kattoi ohjelmointiprojektista kaksi ensimmäistä vaihetta. Kuvassa 15 on kuvattu ohjelmointiprojekti RUP –mallilla. RUP (Rational Unified Process) on iteratiivinen ohjelmistokehityksen prosessimalli. Kukin RUP-vaiheistus (aloitus, tarkennus, rakennus, käyttöönotto) sisältää iteraatioita, jotka koostuvat vesiputousmallin mukaisista vaiheista (määrittely, suunnittelu, toteutus, testaus). Toisin kuin vesiputousmallissa, iteratiivisessa mallissa kehityssyklit ovat lyhyitä, esim. muutama viikko. Iteraatioiden avulla pyritään siihen, että palautetta tehtyjen ratkaisujen toimivuudesta saadaan nopeasti, varhaisessa vaiheessa kehityssykliä. (Haikala & Märijärvi 2004, 44-47, Haikala & Mikkonen 2011, 40-46.)



Kuva 15. Ohjelmointiprojekti kuvattuna RUP –mallilla. Vihreä kehys sisältää toteutuneet vaiheet. Mukaillen Haikala 2011, 46.

Koska aukioloaikasovelluksessa pääpaino on grafiikan sijaan toiminnallisuudessa, on kustannustehokkuuden ja laiteriippumattomuuden vuoksi harkittava, voidaanko aukioloaikasovellusta julkaista web-sovelluksena. Web-sovellusta suunniteltaessa on huomioitava ympäristön rajoitukset, esim. käyttöliittymässä ei voida toteuttaa samoja toiminnallisuuksia kuin natiivisovelluksessa. Web-sovellusta varten tulisikin suunnitella uusi käyttöliittymä. Suunnittelussa voidaan hyödyntää käyttäjätestauksessa saatua palautetta.

LÄHTEET

Amatya, Suyesh 2013. Cross-Platform Mobile Development: An Alternative to Native Mobile Development. Linnaeus University, Department of Computer Science. Master's Thesis. Viitattu 14.12.2014 http://lnu.diva-portal.org/smash/get/diva2:664680/FULLTEXT01.pdf

Apple 2015. Start Developing iOS Apps Today. Using Design Patterns. Viitattu 26.4.2015 https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesignPatterns.html

Apple 2014a. iOS Human Interface Guidelines. Viitattu 5.10.2014 https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/.

Apple 2014b. Datecell. Sovellus. Viitattu 14.4.2015 https://developer.apple.com/library/prerelease/ios/samplecode/DateCell/Introduction/Intro.html

Apple 2013. Table View Programming Guide for iOS. Viitattu 10.4.2015. https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/TableView_iPhone/AboutTableViewsiPhone/AboutTableViewsiPhone.html#//apple_ref/doc/uid/TP40007451

Apple 2012a. View Controller Programming Guide for iOS. Viitattu 18.12.2014 https://developer.apple.com/library/ios/featuredarticles/ViewControllerPGforiPhoneOS/Introduction/Introduction.html

Apple 2012b. NSIndexPath UIKit Additions. Viitattu 11.4.2015 https://developer.apple.com/library/ios/documentation/UIKit/Reference/NSIndexPath_UIKitAdditions/index.html#//apple_ref/doc/uid/TP40007175

Charland, A. & Leroux, B. 2011. Mobile application development: web vs. native. Communications of the ACM 54(5), 49-53. Viitattu 14.4.2015 doi:10.1145/1941487.1941504

Cox, N. & Jones, N. & Szumski, J. 2012. Professional iOS network programming : connecting the enterprise to the iPhone and iPad.Indianapolis, Ind. : Wiley Pub., Inc. ProQuest ebrary. Viitattu 2.5.2015 http://ez.lapinamk.fi:2048/login?url=http://site.ebrary.com/lib/ramklibrary/Doc?id=10613135

Feiler, J. 2013. Treehouse Book Series : iOS 6 Foundations. 3rd Edition. Somerset, NJ, USA: John Wiley & Sons. ProQuest ebrary. Viitattu 11.4.2015 http://ez.lapinamk.fi:2054/lib/ramklibrary/reader.action?ppg=251&docID=10692140&tm=1428724687317

Gartner 2014. Gartner Says By 2018, More Than 50 Percent of Users Will Use a Tablet or Smartphone First for All Online Activities. Newsroom 8.12.2014. Viitattu 13.12.2014 http://www.gartner.com/newsroom/id/2939217.

Haikala, I. & Mikkonen T. 2011. Ohjelmistotuotannon käytännöt. 12. uudistettu painos. Helsinki: Talentum.

Haikala, I. & Märijärvi, J. 2004. Ohjelmistotuotanto. 10. uudistettu painos. Helsinki: Talentum.

Helsingin kaupunginkirjasto 2013. Kirjastokortti kännykässä. Juttuja e-kirjastosta 8.10.2013. Viitattu 13.12.2014 http://www.helmet.fi/fi-FI/Ekirjasto/Juttuja_ekirjastosta/Kirjastokortti_kannykassa%2824641%29

Khan, A. 2015. Objective-C and iOS programming. A simplified approach to developing apps for the Apple iPhone & iPad. Boston, MA: Cengage Learning.

Koskenniemi, Y. 2012. Ketterät menetelmät ohjelmistokehityksessä. Teoksessa P. Hanni-Vaara & O. Kähkönen & M. Merivirta (toim.). Lappilaiset sähköisen liiketoiminnan aallonharjalla: uusia mahdollisuuksia ammatilliseen opetus- ja kehittämistyöhön. Rovaniemi: Rovaniemen ammattikorkeakoulu, 79 - 82. Viitattu 13.12.2014 http://ka.ramk.fi/eJulkaisut/C33_Lappilaiset_sahkoisen_liiketoiminnan_aallonharjalla/

Mikkonen, T. & Taivalsaari, A. 2011. Apps vs. Open Web: The Battle of the Decade. Proceedings of the 2nd Workshop on Software Engineering for Mobile Application Development MSE'2011, 22-26. Viitattu 13.12.2014 http://www.mobileseworkshop.org/papers/6-Mikkonen_Taivalsaari.pdf

Pruikkonen, A. 2014. Katsaus virtuaalikampukseen. Viikon e-Vinkki 15.9.2014. Viitattu 13.12.2014 http://some.lappia.fi/blogs/evinkki/2014/09/15/katsaus-virtuaalikampukseen/.

Rajapakse, D. 2008. Techniques for de-fragmenting mobile applications: A taxonomy. 20th International Conference on Software Engineering and Knowledge Engineering. Viitattu 14.12.2013 https://www.iscs.nus.edu.sg/~damithch/files/SEKE2008.pdf

Rintala, M. & Jokinen, J. 2013. Olioiden ohjelmointi C++:lla. Viitattu 26.4.2015 http://www.cs.tut.fi/~oliot/kirja/olioiden-ohjelmointi-uusin.pdf

Risner, Chris 2013. 31 Days of iOS. Day 7: Making Network Requests. Viitattu 2.5.2015 http://chrisrisner.com/31-Days-of-iOS--Day-7%E2%80%93Making-Network-Requests

Schreurs, B. (ed.) 2011. Reviewing the virtual campus phenomenon. The Rise of large-scale e-learning initiatives worldwide. International council for open

and distance education. Heverlee: EuroPACE ivzw. Viitattu 13.12.2014 http://revica.europace.org/Re.ViCa%20Online%20Handbook.pdf.

Suomen virallinen tilasto 2012. Väestön tieto- ja viestintätekniikan käyttö 2012. Internetin käyttö muualla kuin kotona tai työpaikalla. Helsinki: Tilastokeskus. Viitattu 13.12.2014 http://www.stat.fi/til/sutivi/2012/sutivi_2012_2012-11-07_kat_003_fi.html.

LIITTEET

LIITE 1

Käyttöliittymätestauksen haastattelurunko

Aloitus

Onko ollut tilanteita, että olet koittanut asioida kirjastossa, mutta kirjasto on ollut kiinni?

Ovatko kirjaston aukioloajat ja tieto poikkeavista aukioloajoista helposti löydettävissä?

Käyttöliittymä 1

Käyttäisitkö päivämäärävalintaan mieluimmin valintarullaa (DatePicker) vai perinteistä kuukausikalenterinäkymää?

Vaikuttaako ohjelma sellaiselta, että voisit käyttää sitä ilman ohjetta?

Käyttöliittymä 2

Käyttäisitkö päivämäärävalintaan mieluimmin valintarullaa (DatePicker) vai perinteistä kuukausikalenterinäkymää?

Vaikuttaako ohjelma sellaiselta, että voisit käyttää sitä ilman ohjetta?

Käyttöliittymä 3

Vaikuttaako ohjelma sellaiselta, että voisit käyttää sitä ilman ohjetta?

Lopetus

Minkä kolmesta sovellusvaihtoehdosta asentaisit omaan älypuhelimeesi?